BBN Systems and Technologies Corporation

A Subsidiary of Bolt Beranek and Newman Inc.

Report No. 7139

Parrot: The Janus Paraphraser

Dawn M. MacLaughlin

September 1989

Prepared by:

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, Massachusetts 02138

Prepared for:

Defense Advanced Research Projects Agency (DARPA)
1400 Wilson Boulevard
Arlington, VA 22209-2308

Report No. 7139

Parrot: The Janus Paraphraser

Dawn M. MacLaughlin

September 1989

Prepared by:

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, Massachusetts 02138

Prepared for:

Defense Advanced Research Projects Agency (DARPA)
1400 Wilson Boulevard
Arlington, VA 22209-2308

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-85-C-0016. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 7139	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION BBN Systems and Technologies Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) 10 Moulton Street Cambridge, MA 02238	7b. ADDRESS (City, State, and ZIP Code) Department of the Navy Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable) DARPA-ISTO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-C-0016
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Parrot: The Janus Paraphraser

12. PERSONAL AUTHOR(S)
Dawn M. MachLaughlin

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
technical report	FROM _____ TO _____	1989, September	62

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Natural Language, Text generation, paraphrase, Irus, Janus, Spokesman, Parrot

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes Parrot, a system that generates English text from a WML (World Model Language) expression produced by BBN's Irus-II natural language understanding system. The WML represents Irus-II's understanding of its input query. Parrot works with BBN's Spokesman text generation system to generate the paraphrase. Parrot a[illegible]rus-II are integrated as part of the Janus seamless interface system, which allows a user t[illegible]ery the underlying expert and database sytems in English, as well as use menus, graphics, maps, etc. to access information. This report provides both a general description of Parrot (sections 1 through 6) and implementation details for the person who may want to maintain or extend it (the remaining sections).

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

1 INTRODUCTION

Parrot is a system that generates English text from a WML (World Model Language) expression produced by the IRUS-II Natural Language understanding system (Ayuso et. al. 1989; Weischedel et. al. 1987). The WML represents IRUS-II's understanding of its input query. Parrot works with the Spokesman text generation system (Meteer 1989) to generate the paraphrase.

Parrot and IRUS-II are integrated as part of the Janus seamless interface system, which allows the user to query the underlying expert and database systems in English, as well as use menus, graphics, maps, tables, pointing, etc. to access information. In Janus, when paraphrasing mode is on, a paraphrase of IRUS-II's interpretation of the input query is presented to the user, with the option to continue processing the query, abort processing, or try for another interpretation. Parrot has also been used for disambiguation. If IRUS-II finds more than one semantic interpretation (WML) for a parse, a paraphrase for each interpretation appears in a pop-up window, from which the user can select the intended meaning by choosing a paraphrase. Processing of the query then continues, with the chosen interpretation.

This document is intended to provide both a general description of the paraphraser (sections 2 through 6) and implementation details for the person who may want to maintain or extend it (the remainder of the document). Section 2 describes WML, the input to the paraphraser. Section 3 presents some examples of input queries and their paraphrases. Section 4 gives an overview of how the paraphraser works. Section 5 discusses how the paraphraser uses knowledge from the natural language understanding system: IRUS-II. Section 6 details an example of producing a paraphrase. Section 7 documents the domain objects that the paraphraser defines, and Section 8 discusses how these objects are mapped to text structure and to specification. Section 9 describes some text structure templates that have been provided for resolving role relations. Section 10 describes GENERAL-EVENTs and how they are defined. Section 11 discusses the steps that are necessary to extend the paraphraser to cover more queries or a new domain. Finally, Section 12 discusses the paraphraser's performance: its coverage of a test corpus and phenomena that it cannot yet handle.

2 WML: THE INPUT TO PARROT

IRUS-II represents the meaning of English sentences in a higher order intensional logic called the World Model Language (WML). Intensional logic can represent expressions whose value (extension) varies depending on time ("display Frederick's last 3 reports") or is true in some possible world ("suppose Frederick was C3"). The formal syntax and semantics of WML are described in Weischedel et. al. (1987).

In addition to operators such as SETOF, INTENSION, FORALL, and SKOLEM, a WML expression is comprised constants from a *domain model.* The domain model contains information about the concepts and relations between concepts in a specific domain. The NIKL (Moser 1983) knowledge representation formalism is used to represent the domain model in IRUS-II. A *concept* in the domain model can appear in the WML as a unary predicate or a constant. A *role* in the domain model appears as a binary predicate. In the WML shown in figure 2.1, concepts and roles from the domain model are shown in bold: VESSEL, PORT, and HARPOON-CAPABLE-VESSEL are concepts and IN.PLACE and NAMEOF are roles. Weischedel (1989) discusses the integration of an intensional logic and a taxonomic language as a hybrid representation for the semantic meaning of an utterance.

```
(query
 ((intension
   (present
    (intension
     (iota ?JX1
      (lambda (?JX2)
       (power VESSEL)
       (present
        (intension
         (IN.PLACE ?JX2
          (iota ?JX3 PORT (NAMEOF ?JX3 "San Diego"))))))
       (HARPOON-CAPABLE-VESSEL ?JX1)))))
   time world))
```

Figure 2.1: WML for "Which ships that are in San Diego are harpoon capable?"

3 EXAMPLES

This section shows examples of input queries, each followed by its paraphrase. Since Parrot uses only the semantic representation of the input query, not the lexical or syntactic information resulting from the analysis, the paraphrased text is not identical to the input. From a test corpus containing 105 WMLs in the Navy domain, Parrot produces a correct paraphrase for 95% of them.

```
1. What is the CROVL of the Frederick?
   What is the readiness of Frederick?

2. What is the overall readiness of Frederick?
   What is the readiness of Frederick?
```

Examples 1 and 2, although the input queries differ, produce the same semantic interpretation, so the paraphrase is identical.

```
3. Which indian ocean ships are C3?
   Which vessels in IO are c3?
```

Example 3 shows that a premodifier in the input may be expressed as a post modifying prepositional phrase in the paraphrase.

```
4. How many aren't harpoon capable?
   How many of the c3 vessels in IO aren't harpoon capable?
```

The query in example 4 refers to an entity introduced in example 3. The WML for this query has resolved the reference, so the reference is expressed in the paraphrase. This is useful for confirming that IRUS-II found the intended referent.

```
5. List the ships in San Diego.
   List the vessels in San Diego.

6. What was their previous overall readiness?
   What was the previous readiness of the vessels in San Diego?
```

Similarly, example 6 refers to an entity introduced in example 5.

The remaining examples are intended to both show variety between the input and the paraphrase and highlight the capabilities of the paraphraser.

```
7. Did the Frederick have a Monday c4 readiness?
   Did Frederick have a readiness of c4 on Monday?

8. How fast is Frederick?
   What is the speed of Frederick?
```

9. Is Frederick faster than Spica?
   Is the speed of Frederick greater than the speed of Spica?

10. Did the Frederick's readiness change on 25/12/85?
    Did Frederick change readiness on December 25, 1985?

11. List 3 of the 5 harpoon capable ships.
    List three of the five harpoon capable vessels.

12. Which ships in westpac have an equipment resource rating of C3?
    Which vessels in Westpac have an equipment readiness of c3?

13. How many ships that are harpoon capable are there?
    How many vessels which are harpoon capable are there?

14. How many cruisers are deployed that are c3?
    How many cruisers which are c3 are deployed?

15. What is the Frederick's readiness today?
    What is the readiness of Frederick today?

16. The Frederick's readiness was c1 yesterday?
    The readiness of Frederick was c1 yesterday?

17. Where is the Frederick deployed?
    Where is Frederick deployed?

18. Which ships are within 50 miles of Hawaii?
    Which vessels are within 50 miles of Hawaii?

4 OVERVIEW OF PARROT

Parrot works with the Spokesman natural language generation system (Meteer 1989) to produce the paraphrase. Spokesman is composed of two components: the text planner and the linguistic realization component. In general, the text planner selects the information to be communicated (for paraphrasing, the information is contained in the WML so no selection needs to be done), determines the perspectives (e.g., whether a deployment event should be viewed as an event, as in "Frederick was deployed", or as an object, as in "Frederick's deployment") and organization of the information, and maps the information onto the linguistic resources that the language provides (i.e., open class words and syntactic constructions). The linguistic realization component is Mumble-86 (Meteer et. al. 1987). It carries out the text planner's specification to produce the output text, handling the syntactic and morphological decisions and ensuring that the text is grammatical.

Both components use multiple levels of representation, beginning with the WML and then progressing through more linguistic representations to the final text. Each level completely describes the utterance. As each level is being built, it provides constraints and context for further decisions. Each representation also controls the order of the decisions. Figure 4.1 shows these levels of representation. The levels of representation in the text planner are described in detail in Meteer (1989).

Parrot works with Spokesman's text planner to produce first the text structure and then the linguistic specification, which is the input to the linguistic realization component. It does not participate in the linguistic realization process; this process is handled entirely by Mumble.

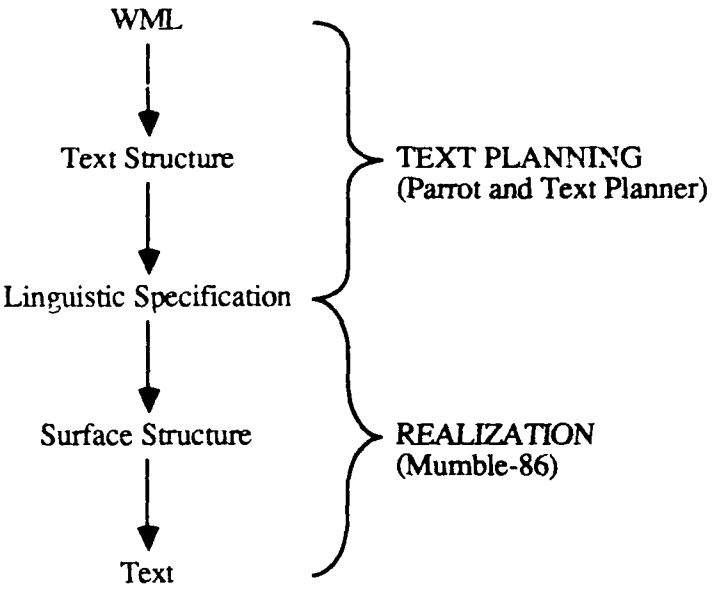


Figure 4.1: Levels of Representation

4.1 From WML to Text Structure

Parrot and the Text Planner work in tandem to create the Text Structure, the central representation of the Text Planner. The Text Structure is a tree structure that represents the constituent structure of the utterance being produced. Every node in the text structure tree represents some constituent in the utterance and contains information about the constituent, such as the WML expression that the consituent represents and the relations between the constituent and other constituents (nodes in the tree). The WML drives the construction of the text structure, but the text structure aids the process by providing a context for making decisions, as the text structure both represents the decisions that have been made so far and constrains the decisions that still need to be made.

Parrot processes a WML expression incrementally, from the outside to the inside, creating some structure to represent the WML and knitting this structure into the tree. The new structure either replaces the contents of the current text structure node or extends the text structure by creating one or more subnodes under the current node. The structure created by Parrot may contain objects defined by the paraphraser, objects defined by the Text Planner, or both. The Text Planner provides a library of objects that the paraphraser can instantiate to represent the semantic type of constituents. Examples are EVENT, RELATION, COMPLEX-OBJECT, INSTANCE-OF-A-KIND, and SAMPLE-OF-A-KIND (for a description of the Text Planner's library of objects, see Meteer

(1989)). These objects commit the constituent to a particular semantic type which is reflected in the syntactic type of the constituent in the final text (e.g., EVENT as a clause, COMPLEX-OBJECT as a noun phrase) and certain grammatical features (e.g., INSTANCE-OF-A-KIND is singular and SAMPLE-OF-A-KIND is plural). Nodes containing Text Planner objects are processed by the Text Planner and nodes containing Parrot objects are processed by Parrot. In this way, the job of building a text structure to represent the WML expression passes back and forth between Parrot and the Text Planner.

The process of building the text structure begins by placing the entire WML into a WML-PARAPHRASE object, which then becomes the contents of the root node of the text structure. The paraphraser is called to process this node, because it contains a paraphraser object. Parrot uses the first item in the WML as an indication of the type of WML expression. It then builds structure appropriate to the expression type, and knits this structure into the text structure tree. Generally, the paraphraser processes only a portion of the WML expression at one time (the outer portion), placing the remainder into a COMPLEX-WML-OBJECT, which becomes part of the structure that is built for the whole expression.

The paraphraser distinguishes between three types of WML expressions: *sentential descriptions, object descriptions,* and *relations*. The following three sections look at each in turn, describing what they are and what text structure results from them.

4.1.1 Sentential Descriptions

Sentential descriptions are queries, assertions, or commands. The WML that is produced by IRUS-II and is the input to Parrot is a sentential description representing the logical meaning of the input sentence. Sentential descriptions are also found embedded in the WML, resulting from an embedded clause in the input. A sentential description is introduced by QUERY, ASSERT, BRING-ABOUT, or a tense such as PRESENT or PAST. The paraphraser builds a QUERY, ASSERTION, or COMMAND object for a sentential description.

4.1.2 Object Descriptions

An object description generally refers to a particular entity. Object descriptions begin with the IOTA or EXISTS operators (IOTA indicates a known entity while EXISTS assumes the existence of an entity). They contain some concept from the domain model, and may or may not have restrictions on that concept. Generally, object descriptions in the WML create COMPLEX OBJECTs

in the text structure. A COMPLEX OBJECT is an object type provided by the Text Planner. It contains a **head**, a set of **restrictions**, and an **underlying object**, which may or may not be different from the head. The head is the "matrix" of the complex object, and the restrictions are "adjuncts." The text planner provides a library of complex object types, such as INSTANCE-OF-A-KIND, SAMPLE-OF-A-KIND, DEFINITE-SET, and GENERIC-MASS. Each type places restrictions on the linguistic specification that is built for the object, e.g., a DEFINITE-SET is a definite plural noun phrase.

Specifically, an object description builds a complex object whose type is derived from various parts of the object description. For instance, a POWER in the object description indicates a plural object, and an IOTA indicates a definite object. The concept in the object description is placed inside a DM-CONCEPT object, which becomes the head of the complex object. If there is a restriction, it is placed in a COMPLEX-WML-OBJECT and added as a restrictive modifier to the complex object. The underlying object is the discourse entity[1] that IRUS-II has built for the object description. Figure 4.2 shows an example of a complex object built from a WML object description.

```
(iota ?X (power VESSEL)
   (IN.PLACE ?X HAWAII)) ------->

#[Definite Set
   head = #[DM Concept VESSEL]
   und obj = <Discourse entity for ?X>
   restrictions = ((restrictive-modifier
                    #[Complex WML Object
                       wml = (IN.PLACE ?X HAWAII)]))]
```

Figure 4.2: WML Object Description creates a Definite Set object

Sometimes it is necessary to handle an object description in some special way. For example, the WML expression below represents "50 knots".

```
(iota ?JX1 MEASUREMENT
  (& (measurement-unit ?JX1 KNOTS)
     (measurement-quantity ?JX1 5)))
```

In this case, we would not want to build a complex object whose head is MEASUREMENT. Instead, we'd like to build an object whose head is KNOTS. To handle this case, a special text structure mapping can be defined for the concept in the object description (MEASUREMENT).

[1] As part of its anaphora resolution mechanism, IRUS-II builds discourse entities for each object description that appears in the WML. Parrot uses these discourse entities as unique underlying application objects in the text structure (see section 5.3).

Parrot will call this mapping on the object description WML instead of building a complex object. Note: the mapping must do all the work, including adding whatever structure it builds into the text structure tree.

Concepts which represent events (e.g., DEPLOYMENT) can be handled through the GENERAL EVENT mechanism discussed in section 10, or by defining a special mapping for the event.

4.1.3 Relations

A *relation* in the WML is a domain model role (two place predicate), with a domain and a range as arguments. Generally, relations are expressed in the paraphrase as either a proposition (a clausal event or state) with the domain and range as its arguments, or as a modifier of either the domain or range. The restrictions in object descriptions are relations. For the purpose of building the text structure, relations can be divided into four categories, according to the relation's role, as shown in figure 4.3.

```
• Special Relations
   (EQUAL <speed> <20 knots>) ------->
   "What is the speed?"
   "the speed is 20 knots"
   "a speed of 20 knots"

• Attribute Relations
   (MAXIMUM-SPEED-OF <vessel> <speed>) ------->
   "the vessel has a maximum speed"
   "the vessel with a maximum speed"
   "the maximum speed of the vessel"

• Mapped Relations
   (FLAG.OF <vessel> "U.S.") ------->
   "the U.S. vessel"

• Event Relations
   (RESUPPLY <Frederick> <Spica>) ------->
   "Frederick is resupplying Spica."
```

Figure 4.3: WML Relations

There are three relations that Parrot treats specially: EQUAL, GREATER-THAN, and LESS-THAN. The ways that the EQUAL relation can be expressed by the paraphraser are shown in figure 4.3. The text structure tree provides the context for choosing among the possible expressions.

Attribute relations are those that are defined as KNACQ (Weischedel et. al. 1989) attributes (see section 5.2). Again, there are several ways to express an attribute relation, and the text structure tree provides the context for choosing the appropriate expression.

Mapped relations are relations that have a particular text structure mapping defined for them (see section 8 for an introduction to mappings). This mapping is to a piece of code that will build the appropriate text structure for the relation, possibly by choosing among several ways to express the relation. Parrot provides a variety of templates that relations may be mapped to, for handling some common cases, e.g., when the range is a modifier of the domain, the domain is a modifier of the range, or the domain and range are related via BE (see section 9).

Event relations are expressed as events. Parrot creates an EVENT[2] object with the role as the event type, the domain as the agent, and the range as the patient to represent the event as a constituent in the text structure.

To build text structure for a relation, Parrot first checks to see if the relation is a mapped relation; that is, if a text structure mapping has been defined for it. If so, then the mapping is instantiated. If not, then Parrot tries to handle it as a special relation, an attribute relation, or an event relation, in that order. The event relation is the default case; that is, if the relation does not fall into the other three categories, an event will be built for it.

4.2 From Text Structure to Linguistic Specification

The second step in generating the paraphrase is building the linguistic specification from the text structure. The linguistic specification must completely specify the information that is to be expressed. It must contain the heads of phrases (the content words), the classes of syntactic structures to be used, and necessary linguistic information like tense and number. Because the text structure tree is largely populated with text planner objects, most of the specification is built by the text planner component itself. Parrot objects are generally found as a constituent head (e.g., the head of an object or the head of an event) and contribute the content word to the phrase, or they are found in a clausal phrase position and contribute the phrase type, but not the head (e.g., QUERY, ASSERTION, and COMMAND objects). If the object is a concept or a role, then the appropriate lexical item is accessed through the domain model. Concepts and roles that represent events, in addition to contributing the lexical head (verb), must have the appropriate additional information

[2] EVENT is an object provided by the Text Planner. It has an event type (usually realized as a verb) and event arguments such as agent, patient, and goal.

defined for them to specify the class of syntactic structures appropriate to the verb and its arguments (e.g., the argument structure class).

4.3 From Linguistic Specification to Text

The linguistic specification that is produced by the Text Planner and Parrot is realized as text by Mumble 86, the linguistic realization component. Parrot treats this phase of the generation as a black box; it must produce the specification, but the specification is realized entirely by Mumble. Therefore, this phase is not discussed in this document (see Meteer, et. al. (1987)).

5 KNOWLEDGE SHARED WITH IRUS-II

Parrot takes advantage of some of the information provided by the reference knowledge modules of IRUS-II (see figure 5.1). It uses the lexicon, the domain model, and the discourse entities that are created by IRUS-II for each object in the WML (every variable that appears in the WML has a corresponding discourse entity). As the figure points out, Parrot does not use the semantic interpretation rules, or Irules.

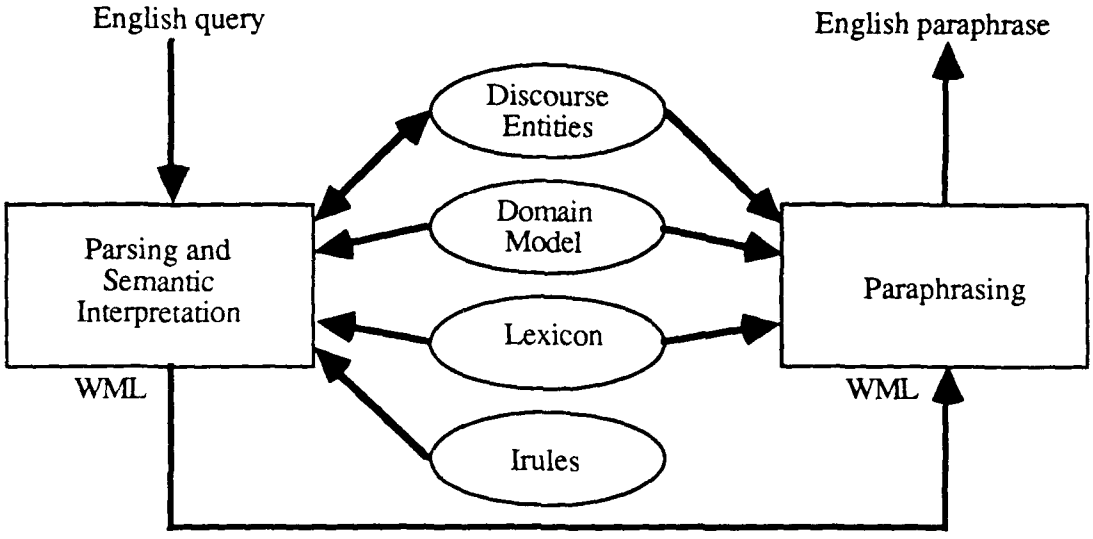


Figure 5.1: Parrot and IRUS-II share resources

5.1 LEXICAL INFORMATION

Lexical information about domain objects can be found in 3 different places.

- A domain model derived via the KNACQ Knowledge Acquisition package contains both lexical and attribute information about domain model objects. Objects in the domain model may have attached to them a set of "vocabulary data," which is a list of lexical items that can be used to refer to the domain model object.

- The semantics field of lexical items in the IRUS-II dictionary refers to objects in the domain model. In the absence of a KNACQ derived domain model, Parrot iterates over the items in the lexicon and uses the semantics field to attach that lexical item to

the appropriate domain model objects, in effect, creating the vocabulary data that would have been supplied by KNACQ.

- Because the order of vocabulary items attached to domain model objects is fairly arbitrary and Parrot must choose a single word to describe the object (currently, Parrot chooses the first one), a table associating domain model objects to lexical items has been provided (this table is called the "preferred word table"). A developer may make an entry in this table when a particular lexical item is desired but is unavailable as the first lexical item from the domain model (alternatively, the domain model can be adjusted to put the lexical items in the desired order, but generally one may want to avoid modifying the domain model for this purpose).

Once a lexical item for a domain model object is found, it is looked up in the IRUS-II lexicon and translated into the type of word object used by Mumble's morphological processor.

5.2 ATTRIBUTE INFORMATION

An attribute relation is a relation between an object and a value. It is analogous to the frame-slot-value relationship in frame language systems and the instance-slot-value relationship in object oriented programming paradigms. In NIKL domain model terms, it is a role which relates one concept to another concept. In a domain model derived via KNACQ, concepts have data attached to them that define the attributes of the concept in terms of roles in the domain model, and the words that invoke those attributes. Parrot uses this information to automatically resolve attribute relations, so that no specific text structure mapping need be written for these relations. An example of an attribute is shown below.

```
(MAX-SPEED-OF <VESSEL> <SPEED>)
```

MAX-SPEED-OF is the name of an attribute on VESSEL, and the value of the attribute is a SPEED. In frame language terms, the domain concept (VESSEL) is the frame, the role (MAX-SPEED-OF) is the slot, and the range concept (SPEED) is the value type.

Parrot may express an attribute in any of the following forms:

- "the ship has a maximum speed"
- "the ship with a maximum speed"
- "the speed of the ship"

The text structure provides the context that allows the paraphraser to choose among these possibililties. See section 6.1, step 7 for an example of this choice.

5.3 Discourse Entities

Parrot uses the discourse entities that are built for concepts as unique, underlying application objects (see Ayuso (1989) for a discussion of IRUS-II's discourse component). A unique underlying object is needed when it is important to know if this exact object has been mentioned already, such as for subsequent reference strategies (pronominalization, definiteness) and syntactic operations (gapping).

6 DETAILED EXAMPLE

This section discusses in detail the first two steps in generating a paraphrase: building the text structure from a WML and building the linguistic specification from the text structure. These are the steps that involve Parrot. The example to be used is the WML shown below, which results from "List the speeds of the ships in the Indian Ocean". The items in bold indicate names of domain model objects.

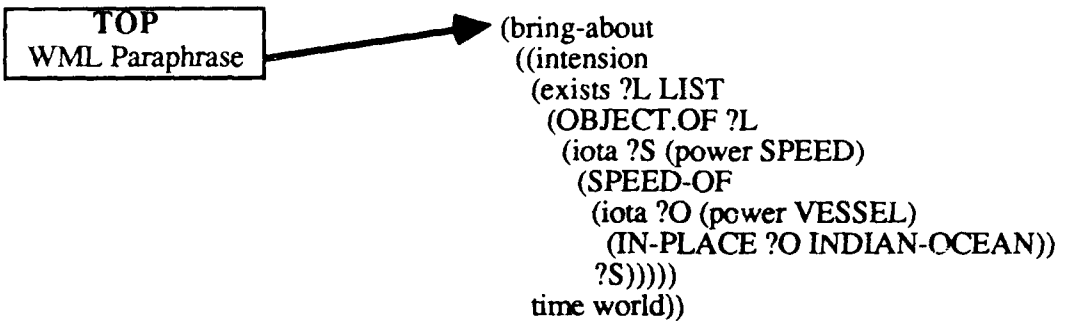
```
(bring-about
  ((intension
    (exists ?L LIST
      (OBJECT.OF ?L
        (iota ?S (power SPEED)
          (SPEED-OF
            (iota ?O (power VESSEL)
              (IN-PLACE ?O INDIAN-OCEAN))
            ?S)))))
    time world))
```

6.1 WML TO TEXT STRUCTURE

The first step in generating the paraphrase is creating the text structure. This section traces through each step of the process for the given example. Each step is accompanied by a figure showing the resulting text structure. The nodes in the text structure are drawn as boxes. The label in bold describes the relation between the node and its parent; the remaining label describes the constituent that the node represents (the CONTENTS of the node).

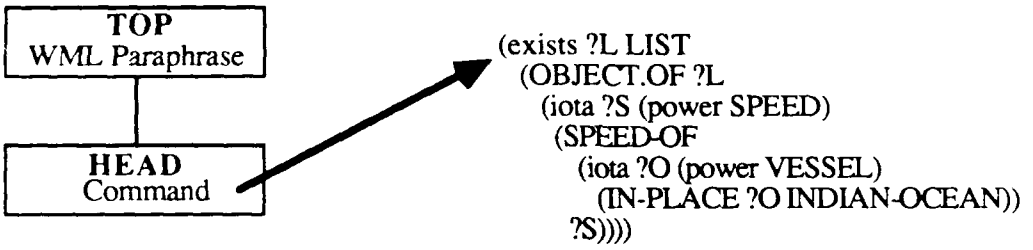
Step 1: Building the root object

The first step is to build the object that goes into the root node of the tree. Parrot creates an object of type WML-PARAPHRASE that contains the input WML and places this object in the root node.



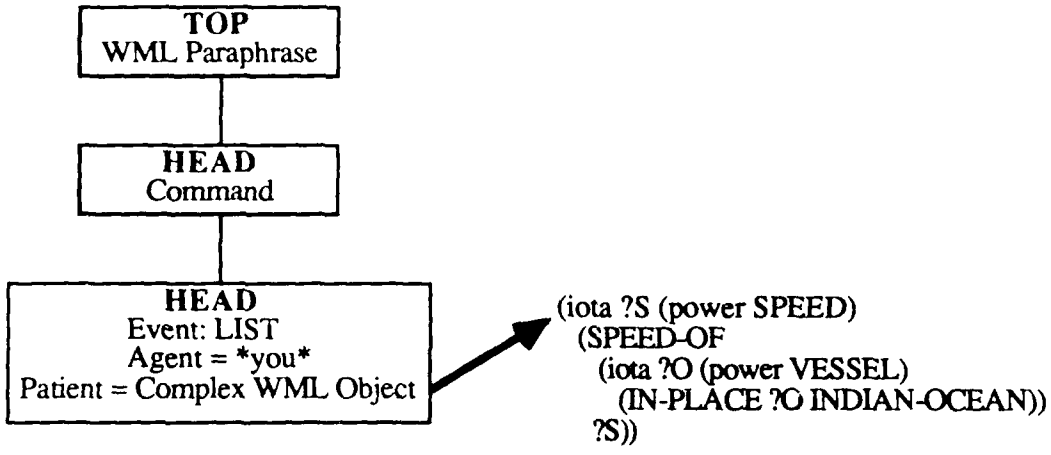
Step 2: Processing the WML-PARAPHRASE object

The WML-PARAPHRASE object uses the text structure mapping for COMPLEX-WML-OBJECT. This mapping looks at the WML inside the object, sees that it is a command WML (the first item in the WML, BRING-ABOUT, indicates its type), and builds a COMMAND object. The outermost layers of the WML (including the intension operator) are stripped off and the remainder is placed inside the COMMAND object. The COMMAND object is added as a subnode of the current node.



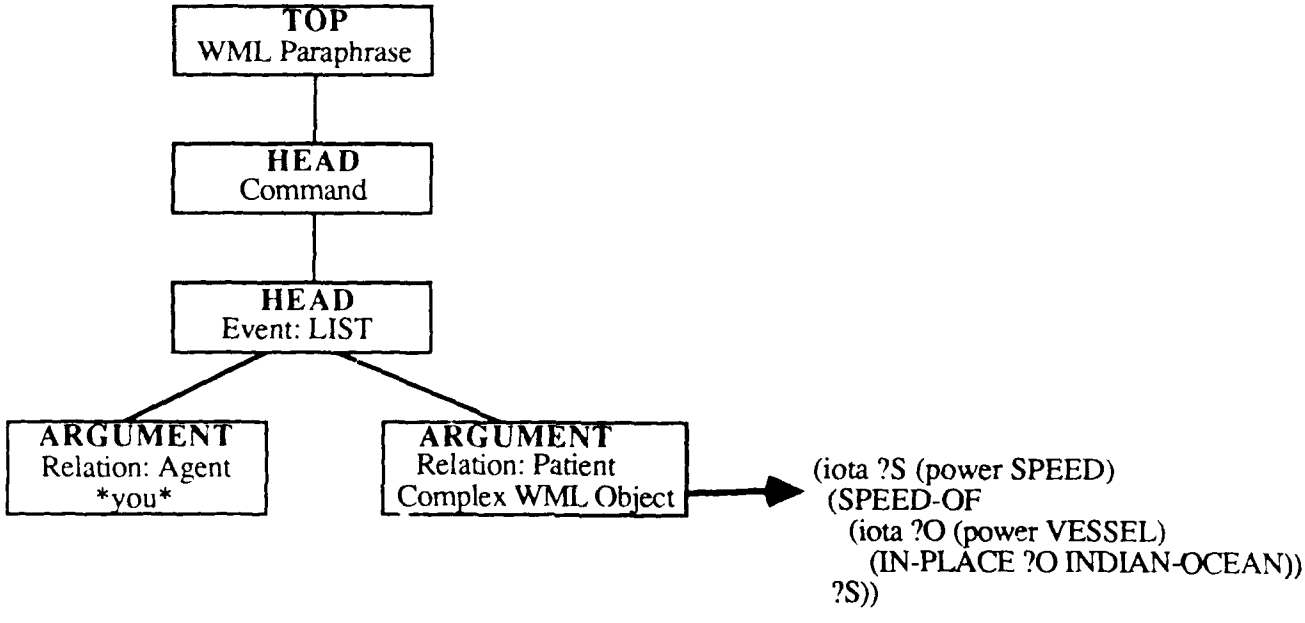
Step 3: Processing the COMMAND object

The text structure mapping for COMMAND looks for the command type (**LIST**) and the object of the command (the range of the **OBJECT.OF** role: (iota ?S (power SPEED) ...)). It builds an EVENT object (a text planner object) whose event is the command type, agent is the global *you*, and patient is a COMPLEX-WML-OBJECT containing the object of the command. The EVENT object is added as a subnode of the current node.



Step 4: Processing the EVENT object

EVENT objects are handled by the Text Planner, not by Parrot. The mapping for EVENTs basically adds subnodes for each of its arguments. The resulting structure is shown below.



Step 5: Processing the EVENT PATIENT

The object which is the event agent (*you*) requires no further processing. It has no mapping so will not build any further structure. It is a leaf node.

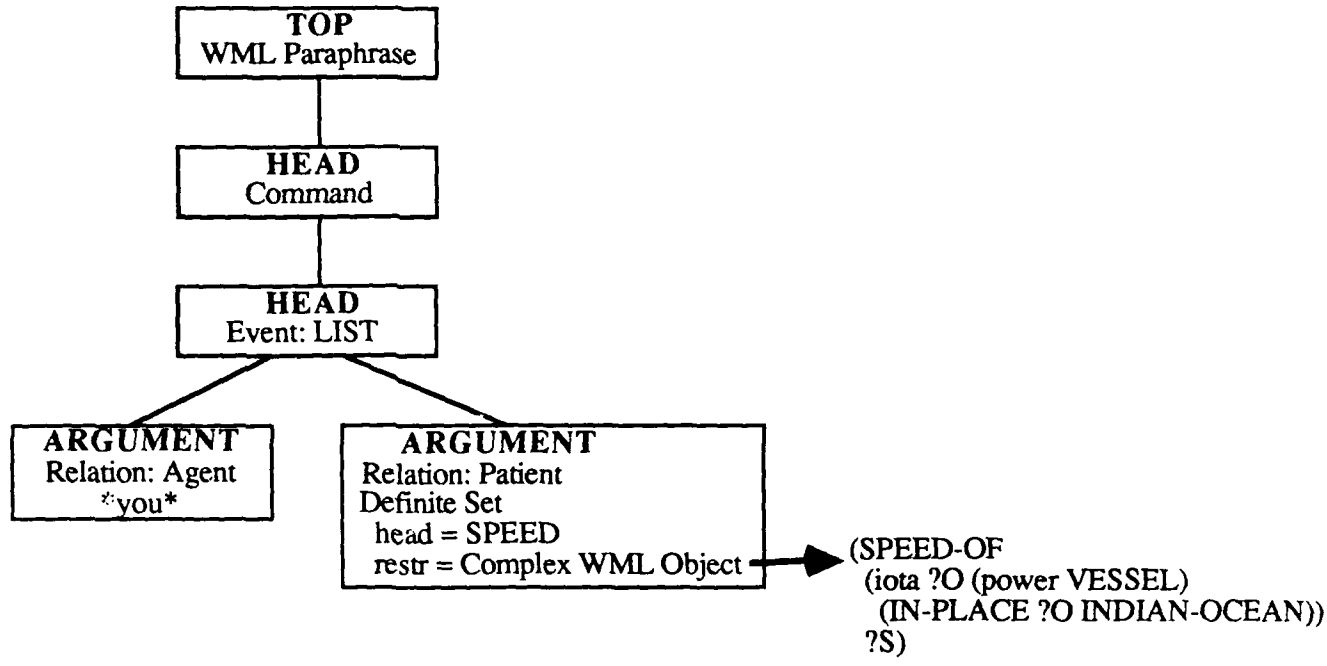
The EVENT patient is a COMPLEX-WML-OBJECT whose WML represents a term. A *term* can be a constant (number or string), a variable, an individual concept, or an object description beginning with IOTA (representing a definite object). The last is the case here.

The template for COMPLEX-WML-OBJECT dispatches to a special function for terms: **build-term**. This function, in turn, calls **build-definite-term**, which handles the IOTA terms. Build-definite-term first looks for a specific mapping for the concept **SPEED**[3].

Since no specific mapping is defined for **SPEED**, the default is used. Build-definite-term calls **build-one-place-predicate**. This function builds a complex object whose object is a DM-CONCEPT containing the concept named **SPEED** and whose underlying application object is the appropriate discourse entity for the term (the discourse entity is accessed by the variable - in this case by ?S). The specific type of complex object that is built is determined by different portions of the term. The IOTA indicates that the object is definite, and the POWER indicates that the object is plural, so a DEFINITE-SET is built. The restriction (speed-of ...) is placed into a COMPLEX-WML-OBJECT, which becomes a restrictive-modifier restriction in the definite set. Also, the variable (?S) and the definite set are recorded in an association list so that the object may be accessed later, if necessary.

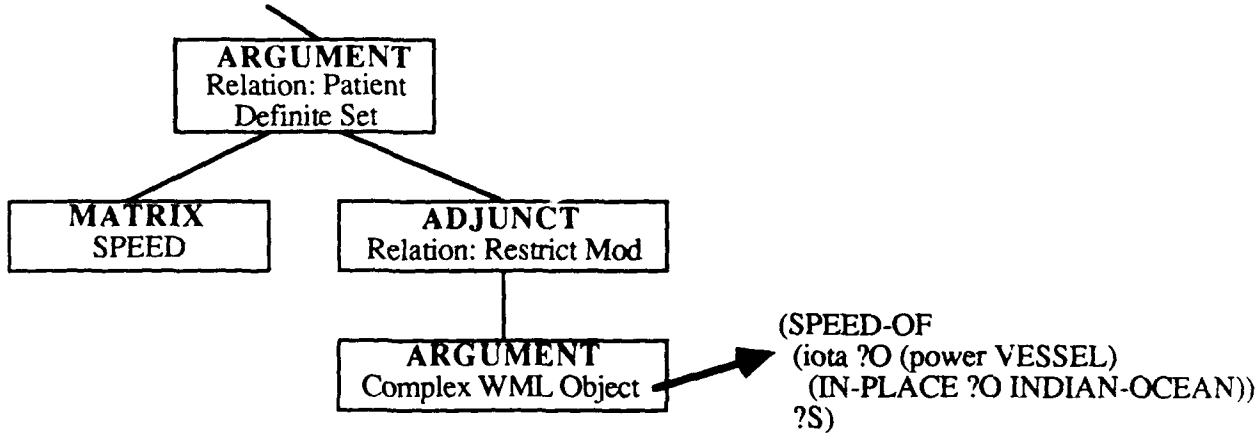
The DEFINITE-SET object replaces the COMPLEX-WML-OBJECT in the current text structure node.

[3] Actually, it looks for a mapping on **:SPEED** - the keyword - to avoid having to deal with different packages.



Step 6: Processing the DEFINITE SET object (and its restriction)

DEFINITE-SET objects are handled by the Text Planner, not by Parrot. The object in the DEFINITE-SET (a *DM-CONCEPT*) is added as a matrix subnode and the SUBORDINATE-RELATION object that was created for the restriction when the DEFINITE-SET was built is added as an adjunct subnode. When the SUBORDINATE-RELATION node is processed, its related object is added as an argument subnode. The figure below shows the text structure after the DEFINITE-SET and its SUBORDINATE-RELATION restriction have been processed.



Step 7: Processing the COMPLEX WML OBJECT for SPEED-OF

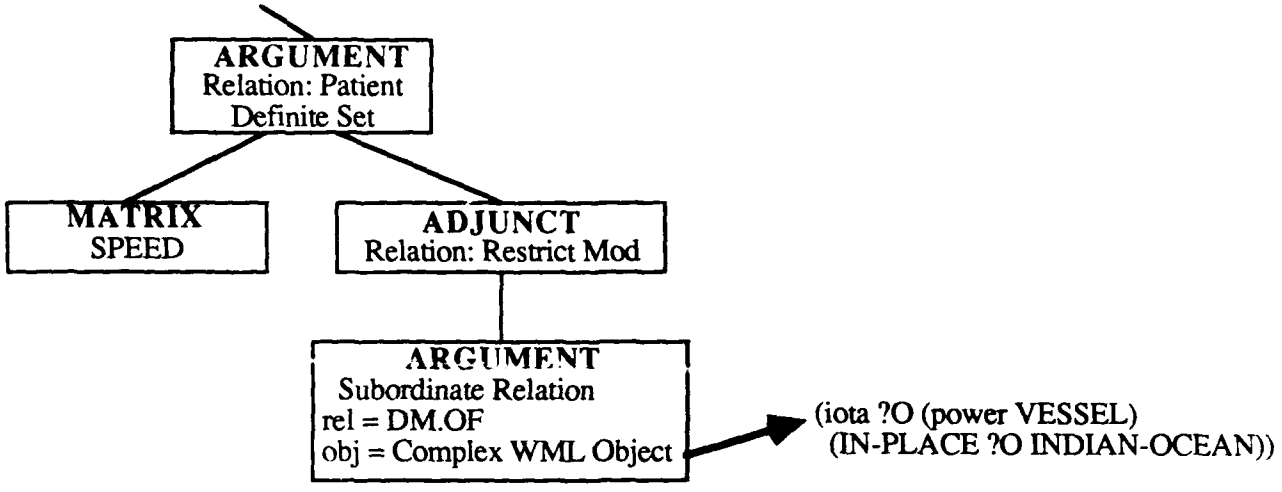
The COMPLEX-WML-OBJECT's WML represents the **SPEED-OF** relation. The template for COMPLEX-WML-OBJECT dispatches to a special function for relations: **build-two-place-predicate**.

The first thing build-two-place-predicate does is see if the role **SPEED-OF** has a special text structure mapping defined for it. No such mapping is found, so the text structure template **build-two-place-predicate-structure** is called with the WML as the argument.

Build-two-place-predicate-structure has several conditions in it to handle the ways in which a relation may be expressed (see section 4.1.3). It finds that **SPEED-OF** is an *attribute* of **VESSEL**. Now it must choose among the three different ways of expressing the attribute:

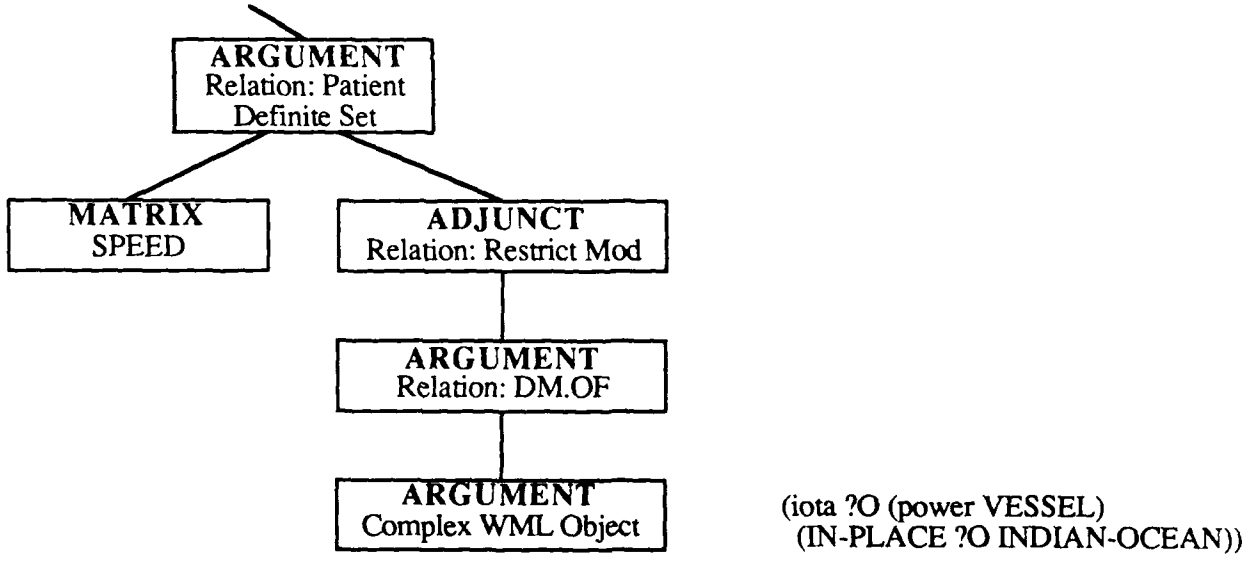
1. As a "have" relation, e.g., "the ship has a speed". This is chosen if the current text structure node contains either an assertion or a query, as both require an event.
2. As a restriction on the domain, e.g., "(the ship) with a speed" (it is assumed that the structure for the text in () is already built, and that the structure for the modifier phrase is what needs to be handled). Parrot chooses this if it sees, by looking up locally in the text structure, if the predicate is modifying the object built for the domain.
3. As a restriction on the range, as in "(the speed) of the ship". Parrot chooses this if it sees, by looking up locally in the text structure, if the predicate is modifying the object built for the range.

In our example, the third choice applies, so Parrot calls the template **domain-is-pp-restriction**, which builds a subordinate relation object: relation = DM.OF, related object = COMPLEX-WML-OBJECT containing the domain portion of the WML - (iota ?O (power vessel) ...). This subordinate relation object replaces the contents in the current text structure node.



Step 8: Processing the SUBORDINATE RELATION object

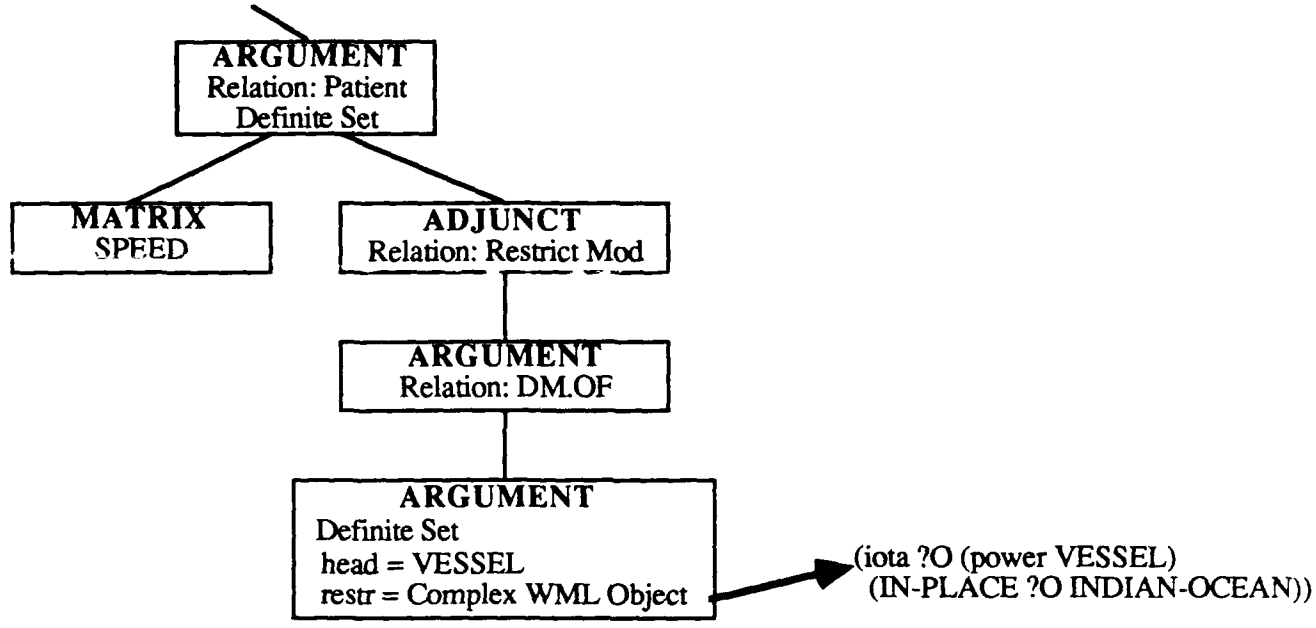
Subordinate relation objects are handled by the Text Planner, not by Parrot. Again, the text structure is expanded and the object of the relation is placed in an argument subnode. The resulting text structure is shown below.



Step 9: Processing the COMPLEX WML OBJECT for VESSEL

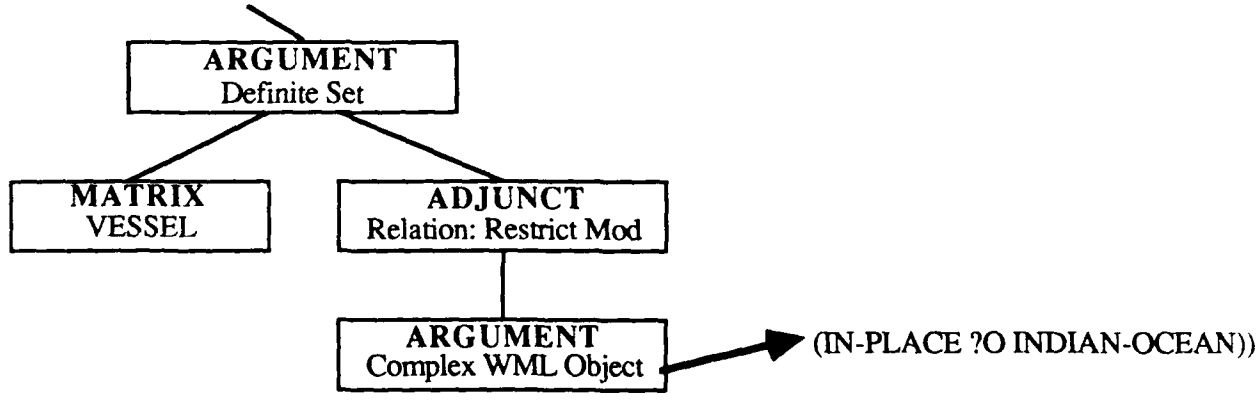
The WML in this COMPLEX-WML-OBJECT is a term, similar to that handled in step 5. There is no default text structure mapping for **VESSEL**, so **build-term** calls **build-one-place-**

predicate, which again builds a DEFINITE-SET object (because of the IOTA and POWER operators). The DEFINITE-SET replaces the COMPLEX-WML-OBJECT in the current text structure node.



Step 10: Processing the DEFINITE SET object

Same as step 6. The resulting structure is shown below.

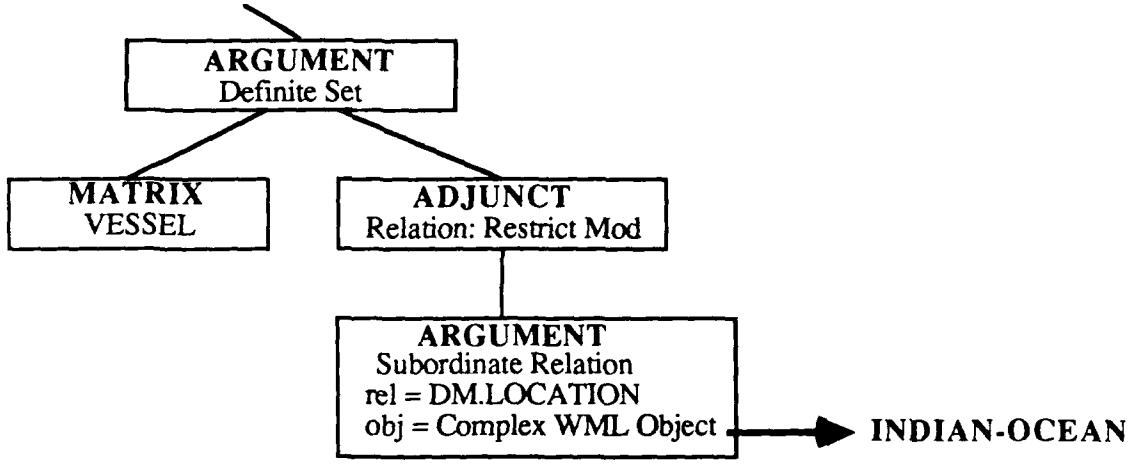


Step 11: Processing the COMPLEX WML OBJECT for IN-PLACE

This WML represents the **IN-PLACE** relation. It is processed similarly to the one handled in step 7, except in this case a specific text structure mapping, **location-structure**, is found for the **IN-PLACE** role. This template chooses between building one of the following structures:

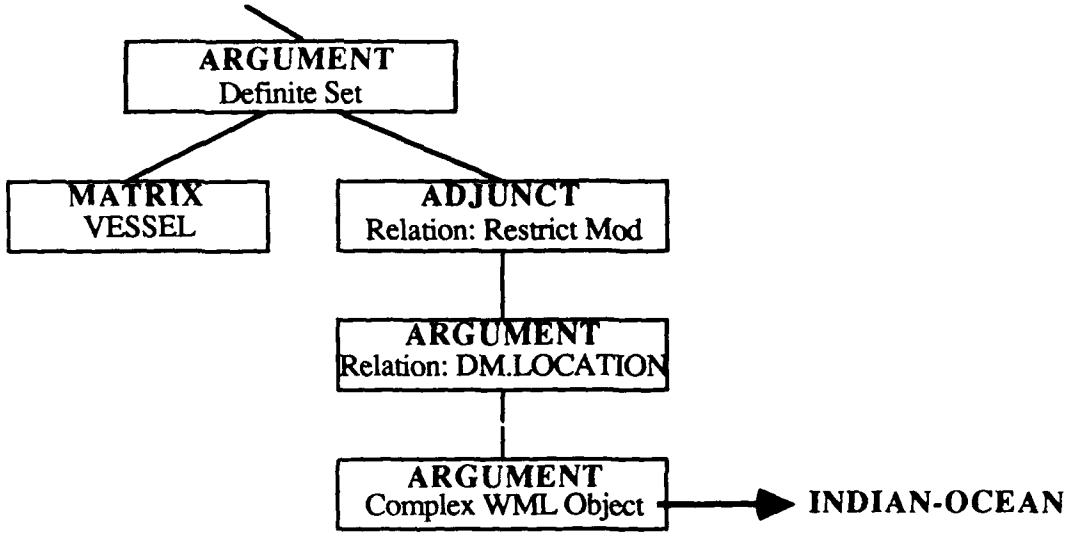
1. domain BE range; e.g., "where is frederick". This special case is chosen if the paraphrase is a query about a location object that is the range of the relation (in this case, the range is a variable that indexes to the "object being queried" (see the description of the QUERY object in section 8).
2. domain as a restriction on the range, related by DM.location; e.g., "(the location) of Frederick". Parrot chooses this if it sees, by looking up locally in the text structure, that the predicate is modifying the object that has been built for the range.
3. range as a restriction on the domain, related by DM.location; e.g., "(the ship) in Hawaii". Parrot chooses this if it sees, by looking up locally in the text structure, that the predicate is modifying the object that has been built for the domain.
4. domain BE DM.location range, e.g., "the ship is in Hawaii". This case is chosen if none of the others apply.

In our example, the 3rd case applies, so a SUBORDINATE-RELATION object whose relation is DM.location and whose related object is a COMPLEX-WML-OBJECT containing the range of the WML (**INDIAN-OCEAN**) is built. This object replaces the contents of the current text structure node.



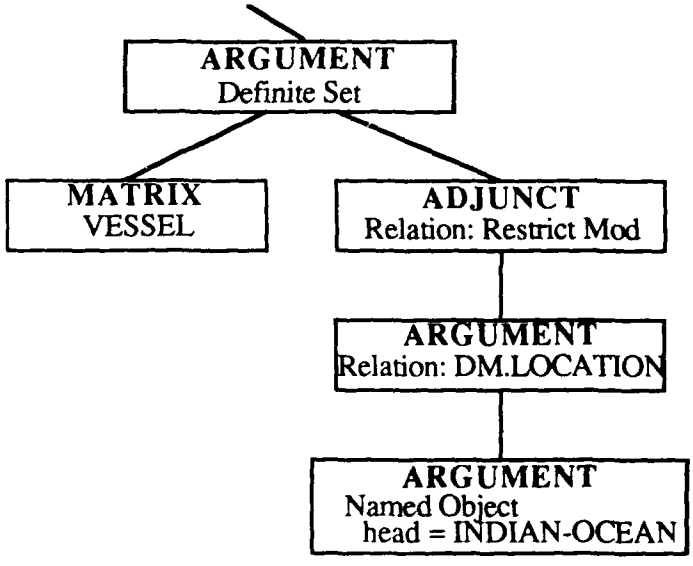
Step 12: Processing the SUBORDINATE_RELATION object

Same as step 8. See the figure below for the resulting text structure.



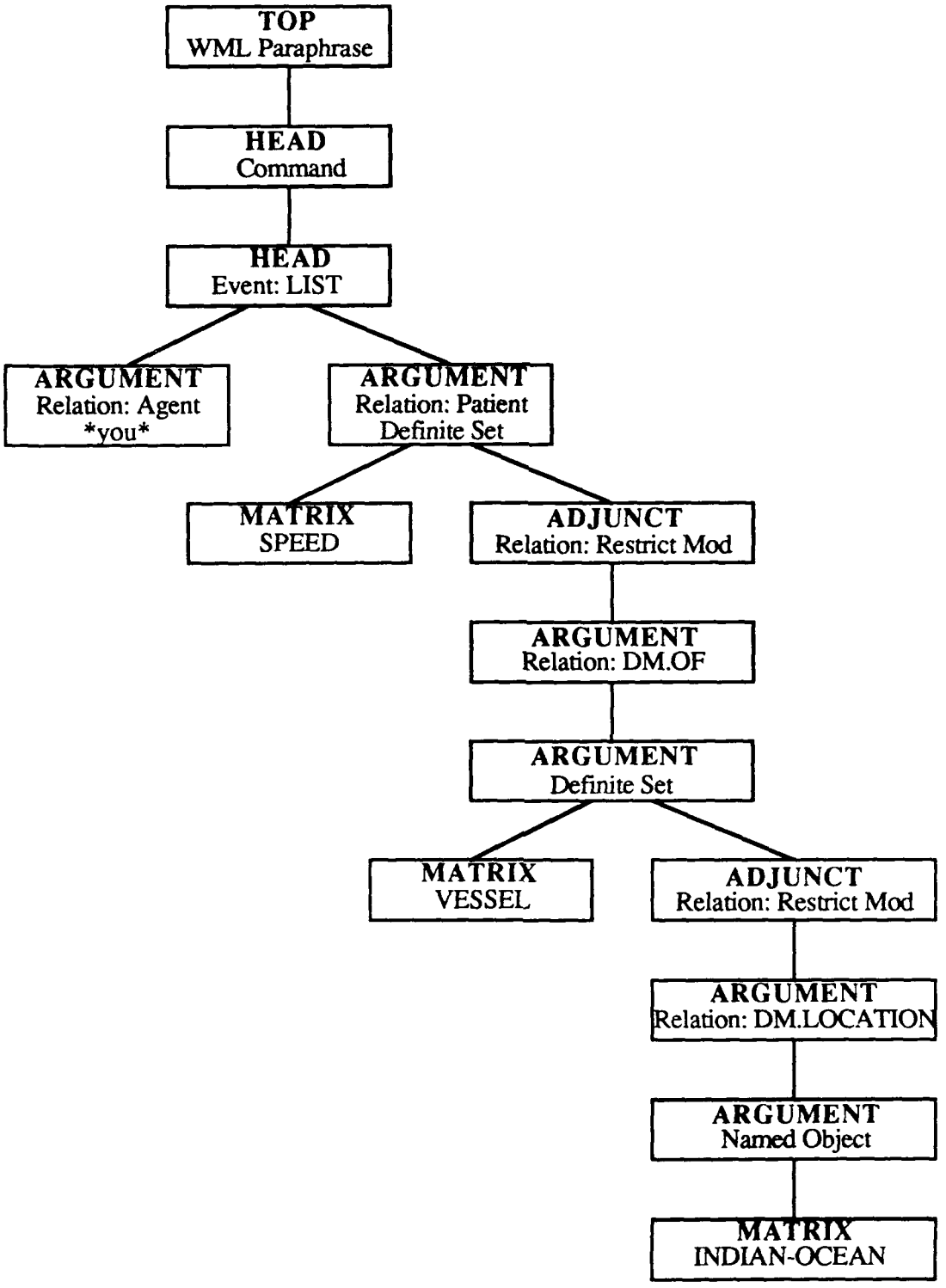
Step 13: Processing the COMPLEX WML OBJECT for INDIAN-OCEAN

The WML in this COMPLEX-WML-OBJECT is simply **INDIAN-OCEAN**: a term which is an individual concept. Build-term still calls build-one-place-predicate, which builds a NAMED-OBJECT: object = DM-CONCEPT containing the concept **INDIAN-OCEAN**, underlying application object = the discourse entity for the individual concept. (Individual concepts don't have variables, but IRUS-II creates discourse entities for them that are accessed via the concept name. A special field in the discourse entity indicates that the entity is for an "individual.") The NAMED-OBJECT replaces the COMPLEX-WML-OBJECT in the text structure.



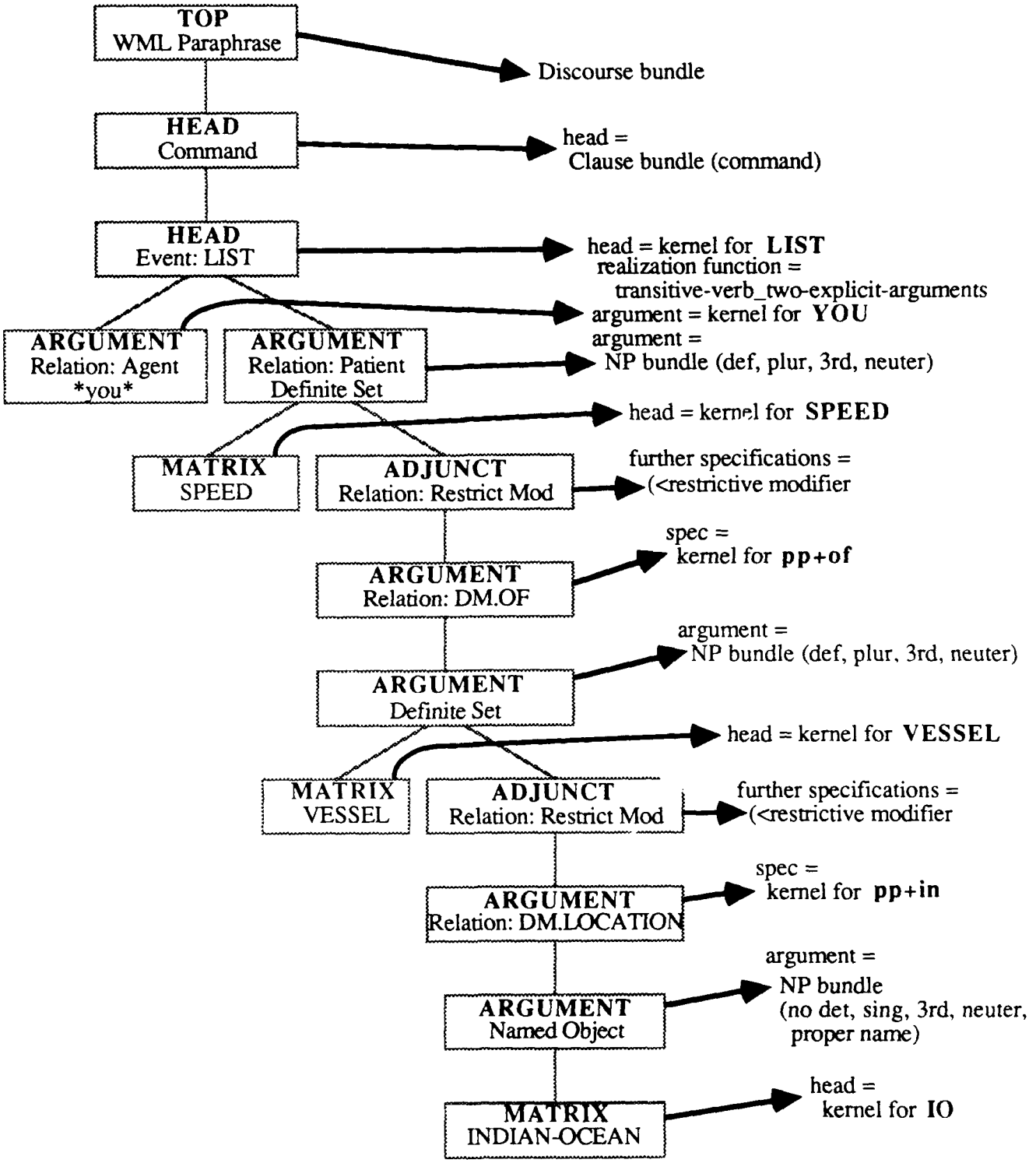
Step 14: Processing the NAMED OBJECT

This object type is handled by the Text Planner, not by Parrot. Similar to the DEFINITE-SET object, the argument is dropped as a matrix subnode (there are no restrictions here). The resulting, and final, text structure is shown below.


TOP
WML Paraphrase
HEAD
Command
HEAD
Event: LIST
ARGUMENT
Relation: Agent
you
ARGUMENT
Relation: Patient
Definite Set
MATRIX
SPEED
ADJUNCT
Relation: Restrict Mod
ARGUMENT
Relation: DM.OF
ARGUMENT
Definite Set
MATRIX
VESSEL
ADJUNCT
Relation: Restrict Mod
ARGUMENT
Relation: DM.LOCATION
ARGUMENT
Named Object
MATRIX
INDIAN-OCEAN

6.2 TEXT STRUCTURE TO SPECIFICATION

The text structure tree is traversed in depth first fashion to build the linguistic specification, which is the input to the realization component: Mumble 86. Each node contributes to the specification. The figure below shows the text structure and resulting specification, with pointers between the text structure nodes and the portions of specification that they contribute. The details of this process and the syntax of the specification language can be found in Meteer (1989).


TOP
WML Paraphrase
Discourse bundle
HEAD
Command
head =
Clause bundle (command)
HEAD
Event: LIST
head = kernel for LIST
realization function =
transitive-verb_two-explicit-arguments
argument = kernel for YOU
ARGUMENT
Relation: Agent
you
ARGUMENT
Relation: Patient
Definite Set
argument =
NP bundle (def, plur, 3rd, neuter)
head = kernel for SPEED
MATRIX
SPEED
ADJUNCT
Relation: Restrict Mod
further specifications =
(<restrictive modifier
spec =
kernel for pp+of
ARGUMENT
Relation: DM.OF
argument =
NP bundle (def, plur, 3rd, neuter)
ARGUMENT
Definite Set
head = kernel for VESSEL
MATRIX
VESSEL
ADJUNCT
Relation: Restrict Mod
further specifications =
(<restrictive modifier
spec =
kernel for pp+in
ARGUMENT
Relation: DM.LOCATION
argument =
NP bundle
(no det, sing, 3rd, neuter,
proper name)
ARGUMENT
Named Object
head =
kernel for IO
MATRIX
INDIAN-OCEAN

7 DOMAIN OBJECTS DEFINED BY PARROT

This section describes the domain objects defined by Parrot. Domain objects are provided to aggregate information from the WML into a manipulable first-class object that specifies both the *type* of the information (e.g., a query, a domain model object) and the information itself (the object that is being queried, the actual object from the domain model). The domain objects are implemented as Common Lisp structures. The structural definitions are presented, followed by a discussion of the object type.

```
(defstruct  (COMPLEX-WML-OBJECT
             (:include message-object)
             (:print "#<Complex-WML-Object>" nil nil))
   wml)
```

This is the basic object for representing a WML expression. The WML field holds the expression that this object represents.

A COMPLEX-WML-OBJECT is created with a WML expression and dropped into the text structure. When a node containing a COMPLEX-WML-OBJECT is processed, the object is replaced by the structure that is built to represent the WML. A COMPLEX-WML-OBJECT should never appear as the contents of a node in the final text structure.

If some other generation component besides Parrot wants to express a WML expression (maybe just a noun phrase description), it can simply create a COMPLEX-WML-OBJECT somewhere in the text structure with that WML expression. When the node containing the COMPLEX-WML-OBJECT is processed, that expression will be handled. This is how Janus's ambiguity resolution mechanism expressed such things as "Do you mean the port Diego Garcia or the vp squadron Diego Garcia?" (Meteer and Shaked 1988). The conjoined phrases in this example originated from WML expressions (specifically, object descriptions), but the overall structure of the phrase was built by the Diagnostic Responses component, not by Parrot.

```
(defstruct  (WML-PARAPHRASE
             (:include complex-wml-object)
             (:print "#<Paraphrase>" nil nil)))
```

This is the top level object that gets inserted into the root node of the text structure tree. It contains the WML expression that is the input to Parrot.

```
(defstruct  (ASSERTION
            (:include complex-wml-object)
            (:print "#<Assertion>" nil nil)))
```

This object is for assertions - both for WML expressions that are assertions (the expression begins with ASSERT) and embedded clauses. The ASSERTION object is similar to the COMPLEX-EVENT text structure object: both commit the constituent semantically to being a complex event and, later, syntactically to being a clause.

```
(defstruct  (COMMAND
            (:include complex-wml-object)
            (:print "#<Command>" nil nil)))
```

This object is for WML expressions that are commands (the expression begins with BRING-ABOUT). It also represents a COMPLEX-EVENT, but the event is a command instead of a proposition (assertion).

```
(defstruct  (QUERY
            (:include complex-wml-object)
            (:print "#<Query>" nil nil)
            (:conc-name query-))
   object-being-queried)
```

This object is for WML expressions that are queries (the expression begins with QUERY). It is similar to the COMPLEX-QUERY text planner object. The object-being-queried is an object of type QUESTION-OBJECT (see below).

```
(defstruct  (QUESTION-OBJECT
            (:print "#<~A ~A>" "Question:"
            (:include message-object)
            (:conc-name qo-))
   object)
```

A question object contains the object that is being questioned in its object slot. This object is some type of complex object built from the object description in the WML expression. The specification mapping on question-object builds the right type of wh pronoun phrase (e.g., who, what, where, which), depending on the type of the object (e.g., person, thing, place, object).

```
(defstruct  (WML-VALUE
             (:include wml-object)
             (:print "#<WML-Value>" nil nil))
   value
   type)
```

This object is for simple values (strings and numbers) that appear as constants in the WML, such as "Frederick" or 10 (in "10th corps"). The type field indicates the type of the value (e.g., ship, identifier). The type may or may not be expressed, depending on the setting of the *say-name-and-type* variable. This variable determines whether you say just "Frederick" or "the ship Frederick" (this can be used, for example, to single out an entity in order to uniquely identify it: "the ship Frederick" vs. "the admiral Frederick"). A WML-VALUE object appears as a leaf node in the text structure.

```
(defstruct  (DM-CONCEPT
             (:conc-name nil)
             (:print "#<~A ~A>" "DM-Concept:"
               (get-dm-object-name dm-concept)))
   concept)
```

This object holds domain model concepts. The concept slot is the actual concept object from the domain model (not the name of the concept). DM-CONCEPT objects appear as leaf nodes in the text structure and contribute a phrasal head to the linguistic specification. The lexical item used to express the concept is looked up in the domain model, as discussed in section 5.1.

```
(defstruct  (DM-ROLE
             (:conc-name nil)
             (:print "#<~A ~A>" "DM-Role:"
               (get-dm-object-name dm-role)))
   role
   range
   domain)
```

This object holds domain model roles. The role slot is the actual role object from the domain model (not the name of the role). The range and domain slots are the pieces of WML that are the range and domain. DM-ROLE objects appear as the event type of an EVENT object (a text planner object). The specification mapping for DM-ROLE looks for a realization function defined for the role name. This realization function specifies the verb and the possible syntactic structures of the clause.

8 DEFAULT MAPPINGS

A mapping is a link between an object type and a set of actions that apply to that object type. When building the text structure, the *text structure mapping* defined for the object in the contents of the text structure node is applied. The text structure mapping links the object type to a *text structure template*, which performs the appropriate actions in extending the text structure. Similarly, the *specification mapping* is accessed while building the linguistic specification. This mapping links the object type to a *specification template*, which maps the object onto the appropriate linguistic resources.

Parrot defines text structure and specification mappings for each of its object types. Text structure mappings are defined for those objects that extend the text structure; they are not defined for objects that appear as leaf nodes (e.g., DM-CONCEPTs). Specification mappings are only defined for those objects that appear in the final text structure.

This section describes the mappings defined for the objects described in section 7. The define-default-mapping form is used establish the mappings (both text structure and specification) from an object type to the appropriate templates. The :class-to-text-structure keyword specifies the text structure mapping - the name of the template and any arguments, and the :class-to-specification-table keyword specifies the specification mapping. The mappings are introduced by giving the define-default-mapping form, followed by a discussion of the templates involved.

```
(define-default-mapping 'COMPLEX-WML-OBJECT
   :class-to-text-structure-table
   (:template-name COMPLEX-WML-OBJECT-STRUCTURE :arguments ()))
```

The text structure template for COMPLEX-WML-OBJECT, COMPLEX-WML-OBJECT-STRUCTURE, is meant to handle any type of WML expression. It dispatches to the appropriate function, depending on the type of the expression, e.g., relation, term, proposition, set, or speech act (assertion, query, or command).

There is no specification mapping for COMPLEX-WML-OBJECT. A COMPLEX-WML-OBJECT should not occur as the final contents of a text structure node; it should always be replaced by some appropriate structure during the construction of the text structure tree.

```
(define-default-mapping 'WML-PARAPHRASE
   :class-to-text-structure-table
   (:template-name COMPLEX-WML-OBJECT-STRUCTURE :arguments ())
   :class-to-specification-table
   (:template-name DISCOURSE-UNIT :arguments ()))
```

The text structure template for WML-PARAPHRASE is the same as that for COMPLEX-WML-OBJECT.

The specification template DISCOURSE-UNIT is provided by the Text Planner. It simply builds a discourse bundle, with the head as yet unspecified.

```
(define-default-mapping 'ASSERTION
   :class-to-text-structure-table
   (:template-name COMPLEX-WML-OBJECT-STRUCTURE :arguments ())
   :class-to-specification-table
   (:template-name EXPRESS-COMPLEX-EVENT :arguments ()))
```

Assertion again uses the same text structure template as COMPLEX-WML-OBJECT.

The specification template EXPRESS-COMPLEX-EVENT is provided by the Text Planner. It builds a clause bundle (the head of the bundle will be contributed by another object) and handles annotations for *tense, speech act, and polarity (negative).*

```
(define-default-mapping 'COMMAND
   :class-to-text-structure-table
   (:template-name COMMAND-OBJECT-STRUCTURE :arguments ())
   :class-to-specification-table
   (:template-name EXPRESS-COMPLEX EVENT :arguments ()))
```

The text structure template for COMMAND is specialized to pick out the command type and the command argument. Commands are assumed to be of the following format: (exists <var> <command-type> (object.of <var> <command argument>)), where the command type is one of the legal *command-types* (e.g., list, display, graph) and the command argument is some object description. The paraphraser can only handle commands of this format (although it is possible to specialize the code to handle some other well-defined command form). The text structure template builds an EVENT object whose event type is the command type, agent is *you*, and patient is the command argument (placed inside a COMPLEX-WML-OBJECT). The EVENT object is placed in a new subnode below the current node, and this node is given a "command" annotation.

The specification template is the same template that ASSERTION uses.

```
(define-default-mapping 'QUERY
   :class-to-text-structure-table
   (:template-name  QUERY-OBJECT-STRUCTURE :arguments ())
   :class-to-specification-table
   (:template-name EXPRESS-COMPLEX-QUERY
    :arguments (  (query-object-being-queried self)
                  (question-type-for-question-object
                     (query-object-being-queried self)))))
```

The text structure template for QUERY builds structure for either a yes-no query or a wh query, depending on the body of the WML expression. The examples in figure 9.1 summarize the different types of queries, with the body shown in bold.

```
1) Yes/No query: "Is Frederick in the Indian Ocean?"
   (query
    ((intension
      (present
       (intension
        (IN.PLACE (iota ?S VESSEL (NAMEOF ?S "Frederick"))
                  INDIAN-OCEAN))))
     time world))

2) Yes/No query: "Has Frederick downgraded?"
   (query
    ((intension
      (past
       (intension
        (exists ?D DOWNGRADE
         (EXPERIENCER ?D (iota ?S VESSEL
                           (NAMEOF ?S "Frederick")))))))))
     time world))

3) Query about existence of an object: "Is there a C1 carrier?"
   (query
    ((intension
      (past
       (intension
        (exists ?S
         (lambda (?Y) CARRIER (READINESS-OF ?Y C1))
         t))))
     time world))

4) WH Query: "Which ships are C4?"
   (query
    ((intension
      (present
       (intension
        (iota ?S (power VESSEL) (READINESS-OF ?S C4)))))
     time world))
```

Figure 8.1: Query Examples

The WML is a yes-no query if either the body of the WML expression is a relation (example 1: "is Frederick in the Indian Ocean"), or it is an EXISTS object description with a restriction, that is,

the restriction is not simply T (example 2: "has Frederick downgraded"). In this case, the template acts as if the WML expression were an assertion (there is no object-being-queried).

Otherwise, the body of the WML expression is an object description, indicating that the question is about some object--a person, a location, a thing, etc. A QUESTION-OBJECT is built as the "object being queried" from the object description. This object is recorded under its variable so that it can be accessed when it is referenced later in the expression. If the query is simply about the existence of the question object (the query is an exists whose restriction is simply T, as in example 3: "is there a C1 carrier"), then an EVENT object is built and added as a subnode to the current text structure node. Otherwise, there is some proposition about the question object (the restriction in the object description) which is placed in a COMPLEX-WML-OBJECT and added as a subnode.

The specification template for QUERY, provided by the Text Planner, is similar to the one for ASSERTION. Both handle annotations for tense and polarity; however, for QUERY, the speech act is always 'query. This template may also add a wh accessory to the clause bundle if there is an object being questioned. The wh accessory is an np bundle that is added either as a simple wh accessory or a wh-adjunct (e.g., for location and time adjuncts), according to the wh annotation[4] on the object being questioned (the first argument that is passed to the template). If the question is about a location, reason, time, person, or thing (specified by the second argument to the template), then the wh accessory is a wh pronoun - where, why, when, who, or what. If the object being questioned is some other thing, like a ship, then the wh accessory is handled in the QUESTION-OBJECT mapping.

```
(define-default-mapping 'QUESTION-OBJECT
   :class-to-text-structure-table
   (:template-name  QUESTION-OBJECT-STRUCTURE :arguments ())
   :class-to-specification-table
   (:template-name WH-FOR-QUESTION-OBJECT :arguments (self))
   :ts-to-specification-final-table
   (:template-name WH-FOR-QUESTION-OBJECT-FINAL :arguments (self)))
```

The text structure template for QUESTION-OBJECT adds a restrictive quantifier subordinate relation to the restrictions on its object; the quantifier is either "which" or "how many". (This has the effect, in the final text, of "which of the ships ...".) It then instantiates the mapping of its object.

[4] The wh annotation is added to the QUESTION-OBJECT by the code that processes the object when it is referred to in the WML. The QUESTION-OBJECT for the adjunct has already been built and recorded under the appropriate Janus variable. When the paraphraser comes across a reference to this variable in the WML, it decides what type of wh annotation to add.

The specification template for QUESTION-OBJECT instantiates the mapping of its object, setting the underlying object of the resulting specification to be the question object (instead of the object inside the QUESTION-OBJECT). The resulting specification is returned.

The final specification template, called during the second pass the text planner makes in building the specification, adds the specification that was built for the QUESTION-OBJECT as the wh accessory to the clause bundle that was built by the QUERY node. This must be done during the second pass for two reasons. First, when you are at the QUERY object, the specification for the QUESTION-OBJECT has not yet been built, so you can not yet add the wh accessory. Second, when you are at the QUESTION-OBJECT (the first time), the specification may not be complete. So you must wait until all the specification has been built before you can add the wh accessory.[5]

```
(define-default-specification-for-a-class 'WML-VALUE
   :template-name  VALUE-KERNEL
   :arguments ((wml-value-value self)))
```

The specification template for WML-VALUE returns a noun kernel containing the value. If the value is a number, it is converted to a string (if the number is less than 10, the number is spelled out - "three"). If the value is not a word known to Mumble, it is automatically defined as a noun.

```
(define-default-specification-for-a-class 'DM-CONCEPT
   :template-name  NP-KERNEL-FOR-DM-CONCEPT
   :arguments (self)))
```

The specification template for DM-CONCEPT finds the word that should be used to express the concept (lexical selection for domain model objects is discussed in section 5.1). A noun kernel containing this word is returned.

```
(define-default-specification-for-a-class 'DM-ROLE
   :template-name  ROLE-SPECIFICATION
   :arguments (self)))
```

The specification template for DM-ROLE first looks to see if a default specification has been defined for the role. If one has, the default specification is instantiated. Otherwise, the template assumes that the role has a word associated with it that has a realization function (DM-ROLES occur

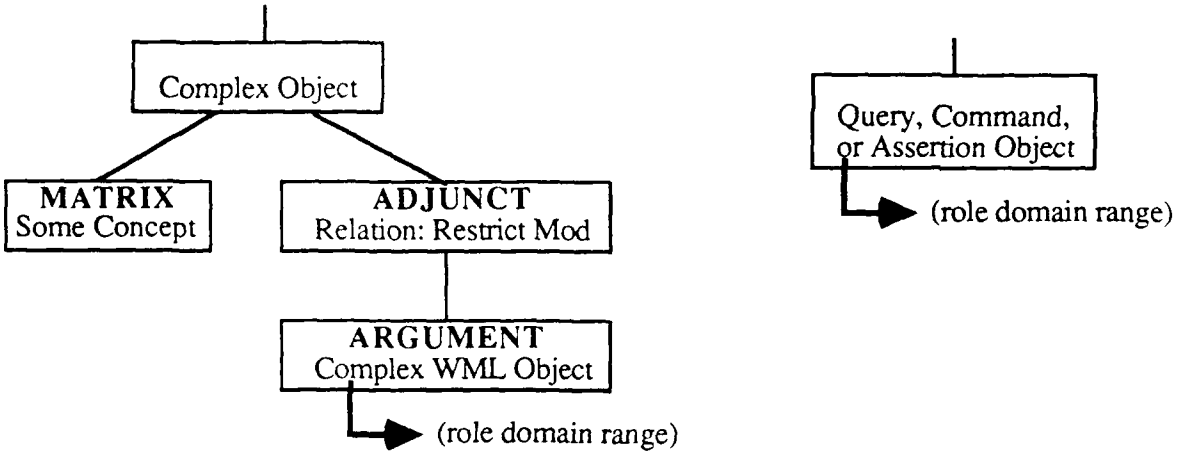
[5] If the accessory could be the object itself, from which Mumble could access the specification for the object, this final mapping would not be needed.

in the text structure as the event of an EVENT object). This realization function specifies the verb of the event clause and the possible syntactic structures.

9 TEXT STRUCTURE TEMPLATES PROVIDED FOR RELATIONS

Some relations can be handled automatically by the paraphraser, namely some special relations (EQUAL, GREATER-THAN, LESS-THAN), attribute relations, and event relations where the role is the event type, the domain is the event agent, and the range is the event patient (section 4.1.3). Relations that do not fall into this category must be individually mapped to a text structure template that will build the desired text structure for expressing the relation. This section describes the text structure templates defined by Parrot that are available to handle some common ways of expressing relations. To use the templates, a text structure mapping linking the role in the relation to the template is defined.

Some templates choose between expressing the relation as a clausal event or as a modifier. The text structure tree provides the context that allows this decision to be made. Relations occur in one of the two contexts shown in figure 9.1, either as a restrictive modifier to some object that is either the domain or the range, as shown in figure 9.1 A, or in an assertion, query, or command context requiring an event, as shown in 9.1 B.



A: Relation as a restriction **B: Relation as a proposition (event)**

Figure 9.1: Text Structure Contexts for Relations

All the templates described here take the relation expression, in the form of (role domain range), as the first argument; and some have additional arguments.

DOMAIN-IS-ADJECTIVE-RESTRICTION *wml*

This template applies to relations that are restrictions on objects. Specifically, the domain is a restriction on the object that is the range. It places the domain of the WML into a COMPLEX-WML-OBJECT. This object then replaces the object in the current text structure node (a COMPLEX-WML-OBJECT containing the WML argument), as shown in figure 9.2.

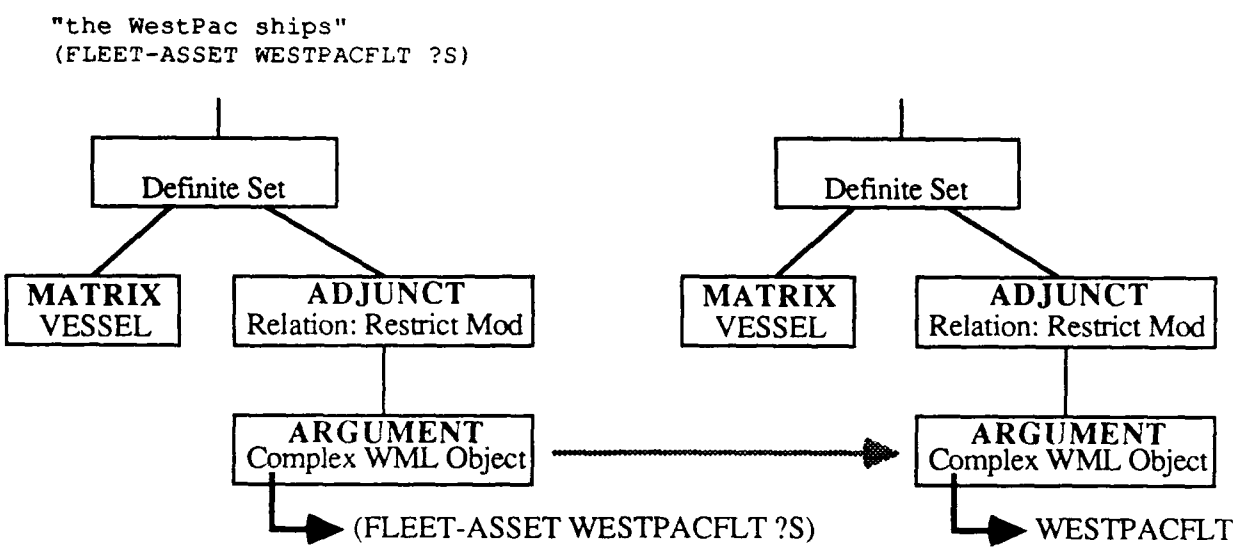


Figure 9.2: Application of Domain-is-Adjective-Restriction template

RANGE-IS-ADJECTIVE-RESTRICTION *wml*

This template is similar to domain-is-adjective-restriction, except the range is a restriction on the domain, instead of the domain being a restriction on the range. Thus, the COMPLEX-WML-OBJECT that replaces the current text structure object contains the range instead of the domain.

DOMAIN-IS-PP-RESTRICTION *wml*

This template is also for relations that are restrictions on objects. It creates a COMPLEX-WML-OBJECT for the domain portion of the WML. A SUBORDINATE-RELATION object (relation = DM.of, object related = new COMPLEX-WML-OBJECT) is created; and this object replaces the contents of the current text structure node (a COMPLEX-WML-OBJECT containing the WML argument).

The difference between this template and the domain-is-adjective-restriction template is that, instead of the domain being a simple modifier of the range, it is related via DM.of, which will be

expressed as a prepositional phrase modifier instead of an adjective modifier. Figure 9.3 shows an example of the application of this template

```
"the readiness of frederick"
(READINESS-OF (iota ?S VESSEL (NAMEOF ?S "Frederick")) ?R)
```

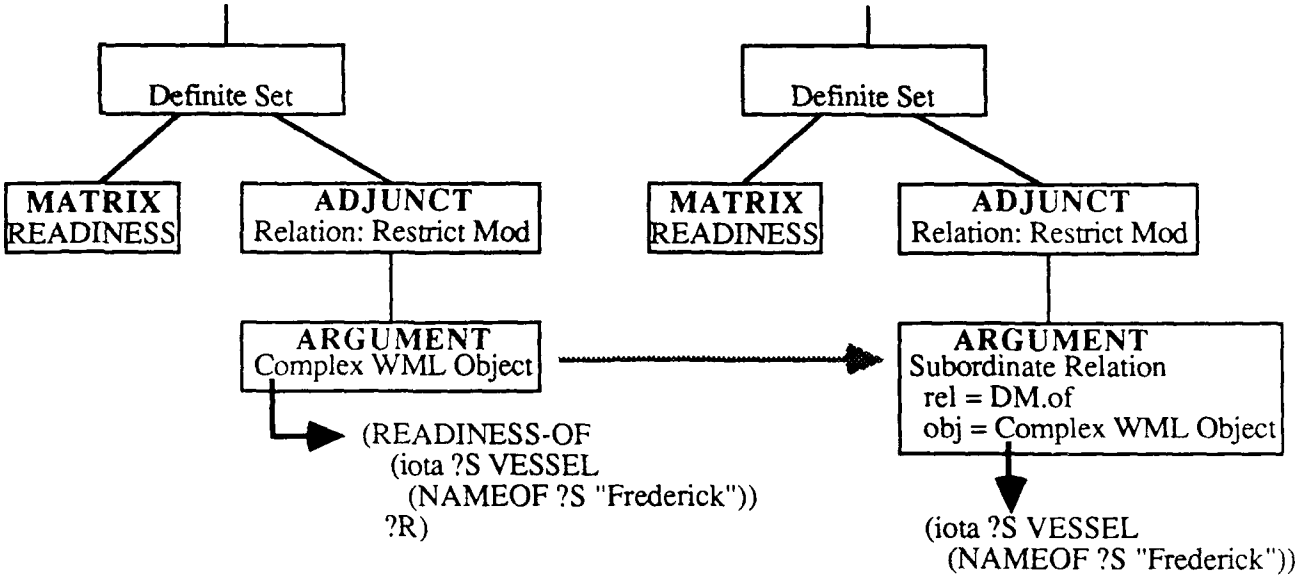


Figure 9.3: Application of Domain-is-PP-Restriction template

RANGE-IS-PP-RESTRICTION *wml relation*

This template is similar to domain-is-pp-restriction, except the range of the WML is used in place of the domain and the relation is a parameter instead of always being DM.of.

DOMAIN-IS-RANGE *wml*

This template applies to relations that are propositions. It builds a BE EVENT, with the domain as the agent (in a COMPLEX-WML-OBJECT) and the range as the patient (also in a COMPLEX-WML-OBJECT). The EVENT is added as a subnode of the current text structure node, as shown in figure 9.4.

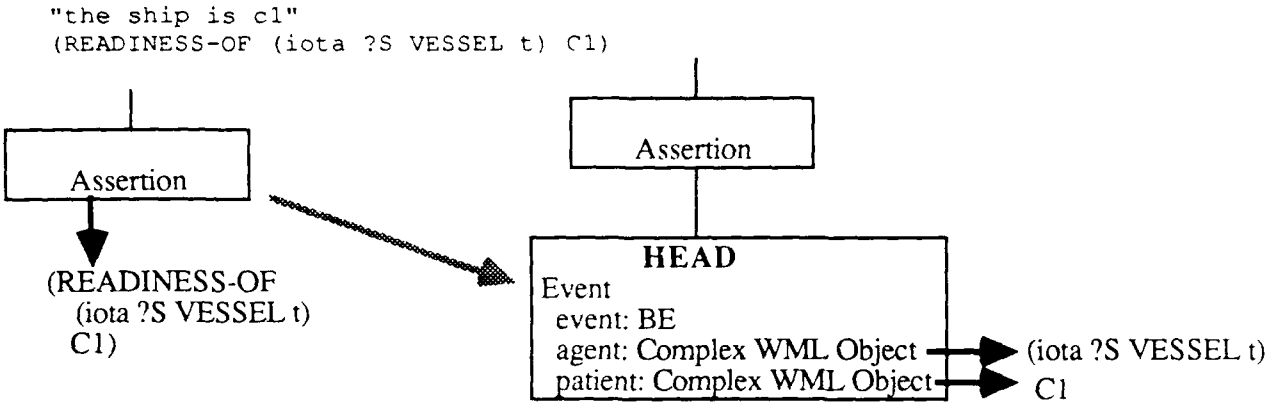


Figure 9.4: Application of Domain-is-Range template

DOMAIN-IS-PP-RANGE *wml pp-relation*

This template is similar to domain-is-range, except the patient of the BE EVENT, instead of being just a COMPLEX-WML-OBJECT containing the range, is a SUBORDINATE-RELATION object (relation = pp-relation argument, related object = COMPLEX-WML-OBJECT containing the range). Figure 9.5 shows an example of the application of this template.

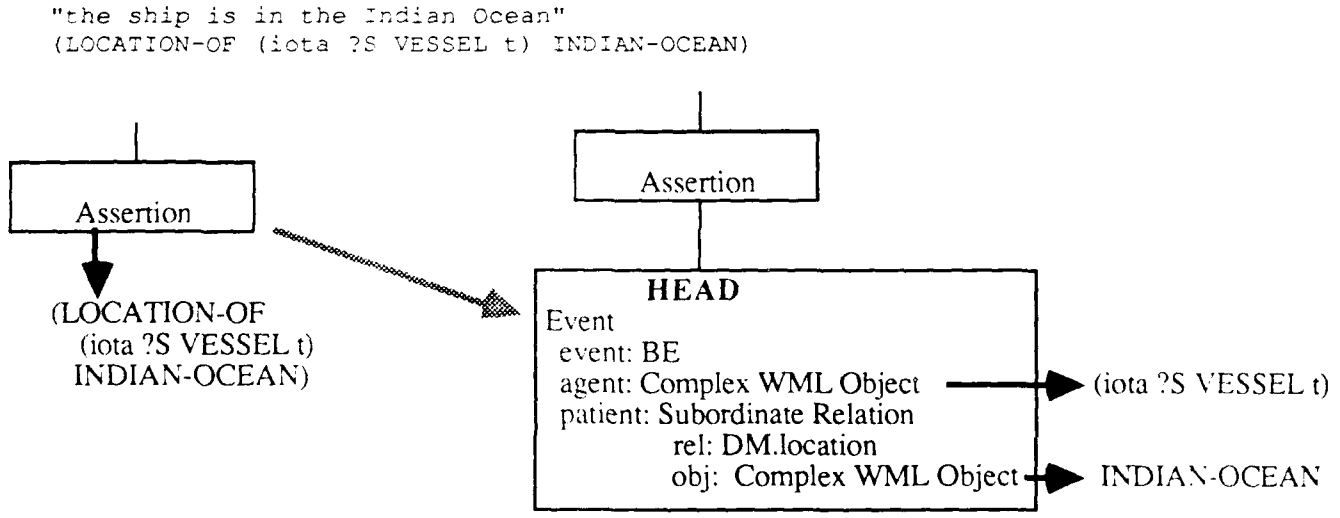


Figure 9.5: Application of Domain-is-PP-Range template

EXPAND-BEPRED-OR-PP-MODIFIER-STRUCTURE *wml relation*

This template chooses between calling either range-is-pp-restriction or domain-is-pp-range ("the ship in Hawaii" vs. "the ship is in Hawaii"). The range-is-pp-restriction template is chosen if the current node is found to be a restriction on an object (the text structure context shown in figure 9.1 A). Otherwise, domain-is-pp-range is called.

Other templates have been defined to handle special but common cases that occur in the WML. Special templates have been defined for **NUMBERP**, a concept that appears for queries like "the number of ships", location predicates, and the **EQUAL**, **GREATER-THAN**, and **LESS-THAN** predicates. These templates are similar to the ones discussed so far (in fact, they call many of these templates), so they are not detailed here.

10 GENERAL EVENTS

Many events are represented as object descriptions in the WML, such as the deployment event shown below (this WML results from "the admiral deployed the ship to the Indian Ocean"):

```
(exists ?JX1 DEPLOYMENT
  (& (AGENT.OF ?JX1 <admiral>)
     (OBJECT.OF ?JX1 <vessel>)
     (DEPLOYED.TO ?JX1 <indian ocean>)))
```

These expressions can be realized in three ways:

1) as an event, with a verb, subject, object, etc, as in this example;

2) as a modifier, as in "the deployed ship"; or

3) as an object, as in "the deployment".

Which expression to use is dictated by the context of the text structure built so far: an event is chosen if the expression is in a QUERY, ASSERTION, or COMMAND object; a modifier is chosen if the expression is a restriction on an object; and an object is chosen otherwise. A general mechanism has been defined to handle these expressions in a uniform manner.

Before describing the mechanism, it is useful to look at how the expression would be handled without it. To paraphrase a WML containing an expression like the DEPLOYMENT event shown above, the developer would have to do the following:

- define a text structure template which would both choose how to express the event and build the appropriate text structure;
- define a specification template which would return the correct argument structure class for the "deploy" verb (for the case where the choice is to express the event as an event clause);
- define a defaut mapping on the DEPLOYMENT concept which would map this concept to the correct text structure and specification templates; and
- define the verb, adjective, and noun forms for "deploy" as words in the Mumble lexicon

The GENERAL EVENT mechanism takes advantages of the regularities in WML exhibited by events such as DEPLOYMENT. Events differ in the arguments and adjuncts that they take, and in the lexical entries that are used to express the event in its three forms. The **define-general-event** macro allows the developer to specify these parameters. The general event definition for DEPLOYMENT is shown below.

```
(define-general-event
 DEPLOYMENT-EVENT
 :DEPLOYMENT
 ( (agent (find-range-of-clause 'irus::AGENT-OF clauses))
   (patient (find-range-of-clause 'irus::OBJECT-OF clauses)) )
 ( (DM.TO (find-range-of-clause 'irus::DEPLOYED-TO clauses) LOCATION))
 "deployed" "deployment" "deploy"
 TRANSITIVE-ERGATIVE-EVENT
 )
```

define-general-event *event concept arguments adjuncts adjective-form noun-form verb-form argument-structure-class*

event This is simply a name that the developer gives to the event, e.g., deployment-event.

concept This is the concept that the event represents (a keyword), which may or may not be the same name as the event argument, e.g., :deployment.

arguments This specifies the arguments to the event. The value of this parameter is a list of pairs of argument name and a form that, when evaluated, will return the WML expression that is the argument. The form is evaluated in a context where the variable CLAUSES is bound to the list of restrictions in the object description that describes the event (with the "&" removed to make a list of clauses instead of an "and" of clauses). Helpful functions to use here are **find-clause**, **find-range-of-clause**, and **find-domain-of-clause**, documented below.

adjuncts This specifies any adjuncts to the event. The value of this parameter is a list of adjunct specifications. Each specification is a list of the relation (e.g., DM.location), a form to evaluate to get the WML expression for the adjunct (same as an argument form), and an optional adjunct type. Currently, adjuncts that are locations must have LOCATION as the adjunct type (locations are treated special in order to handle "where" queries). Other adjunct types will probably be added (e.g., REASON to handle "why" queries).

adjective-form A string which is the adjective form of the event, if any (e.g., "deployed")

noun-form A string which is the noun form of the event, if any (e.g., "deployment")

verb-form A string which is the root verb form of the event (e.g., "deploy")

argument-structure-class The name of an argument structure class defined by the Text Planner. The argument structure class defines the arguments to the verb and the choices for expressing them.

find-clause *first-item clauses*

This function searches the list of clauses for one whose first item matches first-item, returning the clause if it is found.

find-domain-of-clause *first-item clauses*

This function searches the list of clauses for one whose first item matches first-item, returning the domain of the clause (the second item in the clause) if it is found.

find-range-of-clause *first-item clauses*

This function searches the list of clauses for one whose first item matches first-item, returning the range of the clause (the third item in the clause) if it is found.

When a general event is defined, a GENERAL-EVENT object containing the parameters of the event is created and stored in a hash table. A default mapping is defined to map the event to the text structure template and the specification template that handle general events. The lexical items (adjective, noun, and verb) are defined in the Mumble lexicon if necessary (note, if one of the forms has irregularities, it should be defined independently).

The text structure template for general events (**general-event-structure**) decides on what structure to build for the event expression. If the text structure node containing the event is an ASSERTION, QUERY, or COMMAND, then the event is expressed as an event clause. The argument forms are evaluated and collected into COMPLEX-WML-OBJECTs, and an EVENT object is built with these arguments. The EVENT is added as a subnode to the current node. The adjunct forms, if any, are also evaluated and collected into COMPLEX-WML-OBJECTs. These COMPLEX-WML-OBJECTs are then placed into *SUBORDINATE-RELATION objects as the related object*, with the relation being the relation specified for the adjunct. These subordinate relation objects are added as adjunct subnodes.

The event is expressed as a modifier if it occurs in a node that is a restrictive modifier on some object. In this case, a COMPLEX-MODIFIER containing the adjective form for the event is built. This modifier object replaces the contents of the current node.

Finally, the event is expressed as an object if neither of the previous two choices apply. Here, the template treats the event as any other object description; it builds a complex object from the object description. This object replaces the contents of the current text structure node.

The specification template for general events (**general-event-specification**) is only used if the event is being expressed as an EVENT clause. This template chooses the appropriate argument structure, according to the argument structure class that has been specified for the event.

11 EXTENDING PARROT

Unfortunately, all the information the paraphraser needs is not easily available from the NL understanding component. Here is a summary of what you must provide if you are to extend the paraphraser to cover more examples in the Navy domain, or to cover some other domain.

1. Events (e.g., deploy)

Handling a new event type involves defining the following things:

- a Mumble lexical entry for the verb
- a specification template for the verb
- a text structure template to build the apprpriate text structure
- a default mapping for the event type (a concept) that points to the correct text structure and specification templates
- adjective and noun forms for the event

The general event mechanism may be used to define these things (see section 10).

2. Commands (e.g., list, graph)

If the command appears in the WML in the same format as, for example, LIST or DISPLAY, then you just need to

- add the command type to the global list of *commands*
- define the mumble lexical entry for the verb
- define the default realization function for the verb

If the command appears in some complicated WML expression, as may the GRAPH command, then in addition to doing the above, you will need to write special code to build the correct text structure.

3. Relations that are not attributes

Relations need to have mappings defined for them so the paraphraser will know what to do with them (by default, it tries to build an event with the role as the event, the domain as the agent, and the range as the patient - but this is rarely appropriate). Some "all purpose" templates are provided; you just need to define the mappings to tell the paraphraser which one to use (see section 9).

4. Special cases (e.g., measurements, speed, intervals)

New cases always arise, and will require special development.

12 PERFORMANCE

The paraphraser produces paraphrases for approximately 95% of the WMLs produced by IRUS-II (on a test corpus containing 105 wmls, 94 produce paraphrases). However, there are still some areas which are not handled by Parrot, not because of any known constraints imposed by the system, but because time was not available to spend on these areas.

1. Embedded clauses

The only embedded clause that Parrot can produce is a subject relative clause ("the ship that is in the Indian Ocean"). Examples that don't work are: "the snip whose readiness is C1" and "Frederick reported that Spica downgraded".

2. Intervals

"Frederick is five miles from Hawaii" works, but

"Frederick is more than five miles from Hawaii" doesn't work

3. Comparatives and Superlatives

"the fastest ship"

4. WH questions where the object being questioned is moved out of a PP, as in

"What equipment does Frederick have casreps on?"

"What rating has Frederick downgraded to?"

13 REFERENCES

Ayuso, D. M. (1989), "Discourse entities in JANUS", *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada.

Ayuso, D., Donlon, G., MacLaughlin, D., Ramshaw, L., Resnik, P., Shaked, V., and Weischedel, R. (1989), "A Guide to Irus-II Application Development", BBN Report 7144, BBN Systems and Technologies Corporation, Cambridge, MA.

Meteer, Marie W., McDonald, David D., Anderson, Scott D., Forster, David, Gay, Linda S., Huettner, Alison K., and Sibun, Penelope (1987), "Mumble-86: Design and Implementation", COINS Technical Report 87-87, University of Massachusetts, Amherst, MA.

Meteer, Marie and Shaked, Varda (1988), "Strategies for Effective Paraphrasing", *Proceedings of COLING-88*, Budapest, Hungary.

Meteer, Marie (1989), "The Spokesman Natural Language Generation System", BBN Report 7090, BBN Systems and Technologies Corporation, Cambridge, MA.

Moser, M.G. (1983), "An Overview of NIKL, the New Implementation of KL-ONE", *Research in Knowledge Representation for Natural Language Understanding - Annual Report, 1 September 1982 - 31 August 1983*, pages 7-26, BBN Report 5421, BBN Laboratories Incorporated, Cambridge, MA.

Weischedel, Ralph M. (1989), "A Hybrid Approach to Representation in the JANUS Natural Language Processor", *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada.

Weischedel, R., Ayuso, D., Haas, A., Hinrichs, E., Scha, R., Shaked, V., and Stallard, D. (1987), "Research and Development in Natural Language Understanding as Part of the Strategic Computing Program", Annual Technical Report, December 1985 - December 1986, BBN Report 6522, Bolt Beranek and Newman, Cambridge, MA.

Weischedel, R., Bobrow, R., Ayuso, D., and Ramshaw, L. (1989), "Portability in the Janus Natural Language Interface", *Proceedings of Speech and Natural Language*, DARPA workshop, Philadelphia, Pennsylvania.